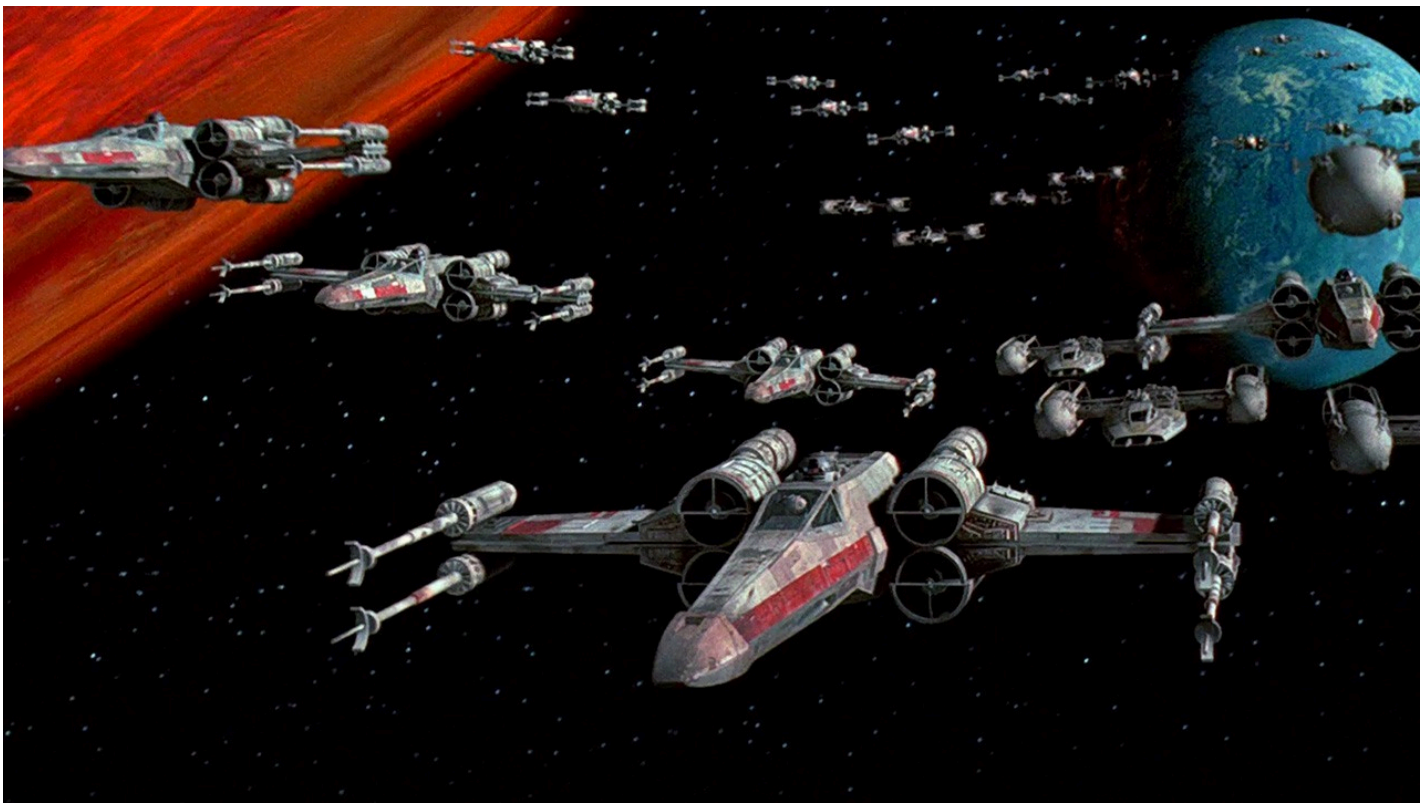
Generalitat de Catalunya
Departament d'Educació
Institut Esteve Terradas i Illa
Bonavista 70
08940 Cornellà de Llobregat
Tel. 93 377 11 00
a8016781@xtec.cat
http://www.esteveterradas.cat

Project 24/25 - Space Wars



Members: Biel Marín
Miguel Angel
Marc Losada
Last Day: 19/05/2025

Generalitat de Catalunya
Departament d'Educació
Institut Esteve Terradas i Illa
Bonavista 70
08940 Cornellà de Llobregat
Tel. 93 377 11 00
a8016781@xtec.cat
http://www.esteveterradas.cat

Index

Generalitat de Catalunya
Departament d'Educació
Institut Esteve Terradas i Illa
Bonavista 70
08940 Cornellà de Llobregat
Tel. 93 377 11 00
a8016781@xtec.cat
http://www.esteveterradas.cat

Project Introduction

What is Space Wars?

Space Wars is a video game based on a popular one called *OGame*. In this game, we live day-to-day on a planet where we have resources, weaponry, and defenses to protect ourselves from attacks by other conquerors.

To survive, we must build defenses to maintain our position against enemy attacks, and also create a fleet capable of intercepting and destroying incoming enemy ships.

To build and prepare for attacks, we need **Metal** and **Deuterium**, the two core materials used for all constructions. Without them, we cannot build anything and risk being defeated. These materials regenerate periodically and can also be gathered from the remains of destroyed ships. If we win a battle, we can store the debris from the conquered planet.

Fleet attacks will be frequent throughout the game, so you can't leave your planet unattended. You must constantly improve and repair everything possible once battles end.
Good luck!

What do we want to achieve?

The goal of this project is to implement everything we've learned throughout the course into a **single mission: building the most complete game possible**.

But what do we mean by "the most complete game possible"?

It's not just about running a playable game. We aim to include:

- An **official website**
- A **database** to store in-game data (battles, planets, fleets, defenses…)
- A **class management system** to structure the game
- And much more.

Generalitat de Catalunya
Departament d'Educació
Institut Esteve Terradas i Illa
Bonavista 70
08940 Cornellà de Llobregat
Tel. 93 377 11 00
a8016781@xtec.cat
http://www.esteveterradas.cat

Tools Used:

For this project, we used:

- **Java** with **Eclipse IDE**: for its ease of object distribution and organization.
- **JFrame (Java Swing)**: for the graphical interface.
- **Blender**: for creating custom 3D models used in the game (especially fleets and defenses).
- **HTML, CSS, JavaScript** (with Visual Studio Code): for the official website.
- **XSL/XML**: to extract game session data for external viewing.
- **StarUML**: to create UML diagrams (used instead of draw.io), very intuitive even for beginners.

Analysis and Design

Functional Specification:

- **In-Game Creation**: As long as the player has resources (Metal and Deuterium), they can upgrade their technology level (attack and defense) and create new defenses and fleets for planet conquests and self-defense.
- **How does a conquest happen?** It happens automatically while the player is in-game. But don't get too relaxed — conquests come quickly, so you always have to be ready for surprises.
- **Battle Report Presentation**: After each battle, a report is generated clearly showing initial units and losses for both sides, resources lost and gained, the winner of the battle, and the current state of the planet.

Non-Functional Specification:

- **Persistent Data Storage**: All relevant game data is stored persistently in a MySQL database on a XAMPP server. This ensures data remains available between sessions for later analysis or reports.
  We implemented this using a Java/SQL management system that separates business logic from data access.
- **Modular Code**: The game uses a modular code structure where each main component (e.g., Planet, Battle, Military) is encapsulated in its own class. This makes maintenance easier and allows new features to be added without changing large portions of the code.

Generalitat de Catalunya
Departament d'Educació
Institut Esteve Terradas i Illa
Bonavista 70
08940 Cornellà de Llobregat
Tel. 93 377 11 00
a8016781@xtec.cat
http://www.esteveterradas.cat

- **Complete Logs**: For each in-game battle, a detailed log is generated and saved to a SQL table. This includes every action line, tracking the evolution of the combat and helping with error diagnostics.

Game Structure:

- **Initial State**: Players must either create an account or log in. Once in, they start with initial resources and fleets/defenses, allowing for early improvements or purchases.
- **Actions**: A menu provides various options, such as buying fleets and upgrading technology. After some time (about a minute), a warning notifies players that someone is about to attack their planet.
- **Battle System**: The game features two types of units: military/fleet (e.g., LightHunter, HeavyHunter…) and defensive (e.g., MissileLauncher, IonCannon…). Each unit has armor (affected by defense tech), damage (boosted by attack tech), and a resource cost (metal + deuterium).
- **Combat Mechanics**: Battles run in automatic turns. Units attack in order, damage is calculated, and casualties are applied immediately. A battle ends when one side has less than 20% of its initial forces remaining.

UML Diagrams:

Class Diagram: This includes the main system classes with their key attributes and functions:

- **Planet:** Manages the planet's actions, resources, and units.
- **MilitaryUnit:** Interface for all military units' functions.
- **PlanetRepository:** Handles SQL data access and operations.
- **Battle:** Controls combat logic like attacks and armor reduction.

Generalitat de Catalunya
Departament d'Educació
Institut Esteve Terradas i Illa
Bonavista 70
08940 Cornellà de Llobregat
Tel. 93 377 11 00
a8016781@xtec.cat
http://www.esteveterradas.cat

Architecture and Development

Source Code Structure:

The project is divided into multiple classes, each corresponding to different components of the game.
The code is organized to ensure:

- **Readability** (even for non-programmers),
- **Modularity**, and
- **Maintainability**.

The structure follows **object-oriented architecture**, grouped under the main package:

```
src/
├── User.java
├── Planet.java
├── PlanetRepository.java
├── Battle.java
├── MilitaryUnit.java
├── LightHunter.java
├── HeavyHunter.java
├── BattleShip.java
├── ArmoredShip.java
├── MissileLauncher.java
├── IonCannon.java
├── PlasmaCannon.java
├── ResourceException.java
├── Variables.java
└── PlanetWars.java (main)

res/
└── img/
    ├── Asset_LightHunter.png
    ├── Asset_HeavyHunter.png
    ├── Asset_ArmoredShip.png
    └── etc…
```

Generalitat de Catalunya
Departament d'Educació
Institut Esteve Terradas i Illa
Bonavista 70
08940 Cornellà de Llobregat
Tel. 93 377 11 00
a8016781@xtec.cat
http://www.esteveterradas.cat

Explanation of Main Components:

User: Represents the player, including their name and password, used for database referencing.
Planet: Manages the player's planet, including resources, technologies, and military units. Enables construction and upgrades.
PlanetRepository: Contains all operations related to the database (MySQL). It's the **persistence layer**, handling:

- Planet creation
- Logging unit builds and losses
- Starting battles
- Saving battle logs
- Retrieving information

MilitaryUnit (Interface): Base interface defining shared properties (damage, armor, probabilities) for all units. Implemented by both military and defense units.
Ship (Abstract Class): Base for all offensive units. Implements the interface to handle:

- Attack probabilities
- Armor reduction
- Resource consumption during creation
  Subclasses: **LightHunter**, **HeavyHunter**, **BattleShip**, **AmoredShip**

Defense (Abstract Class): Base for all defensive units. Same behavior as a ship but for defense. Subclasses: **MissileLauncher**, **IonCannon**, **PlasmaCannon**
Battle: Controls the entire combat system between player and enemy:

- Turn handling
- Damage calculation
- Casualty recording in SQL
- Combat logging
- Generates final battle report

Variables: Stores all **global constants** for the game (base costs, damage, tech multipliers, attack probabilities…).
ResourceException: Custom exception thrown when a player lacks resources to perform an action (e.g., building a unit). If some resources are available, partial construction may proceed until the limit is reached.
PlanetWars (Main): The game's **entry point**. Used for testing and running the main features. Built using **JFrame (graphical interface)** for an intuitive user experience.

Generalitat de Catalunya
Departament d'Educació
Institut Esteve Terradas i Illa
Bonavista 70
08940 Cornellà de Llobregat
Tel. 93 377 11 00
a8016781@xtec.cat
http://www.esteveterradas.cat

Game Data Flow and Event Sequence:

1. **Initialization**
   - The user creates or logs into an account.
   - A planet is created (or retrieved) with:
     - Initial resources
     - Level 1 technologies
     - Starting fleets and defenses
2. **Unit Construction**
   - When a unit is built:
     - System checks for sufficient resources
     - Unit is created in memory
     - Changes are logged in the database (e.g., resource usage, unit count)
3. **Technology Upgrades**
   - If enough **Deuterium** is available, tech level is increased and resources reduced accordingly.
4. **Battle Start**
   - After a certain time, an enemy army spawns.
   - Battle starts automatically (no visual scene).
   - A record is inserted into **battle_stats**, with a detailed log in **battle_log**.
5. **Combat Execution**
   - Units attack in turn-based order.
     Casualties and constructions are updated in **planet_battle_army** or **planet_battle_defense**.
6. **Battle End**
   - The winner is determined by which side has more than **20%** of its initial forces.
   - The losing planet's remains become resources for the victor.

Generalitat de Catalunya
Departament d'Educació
Institut Esteve Terradas i Illa
Bonavista 70
08940 Cornellà de Llobregat
Tel. 93 377 11 00
a8016781@xtec.cat
http://www.esteveterradas.cat

Testing and Debugging

Tests Performed:

Throughout the project, we conducted multiple **manual and functional tests** to validate:

- Proper execution of all game features
- Persistent data handling
- Detection of any data flow errors or major bugs

Examples of test cases:

- ☑ Creating new planets with initial resources and verifying correct insertion into the MySQL database (**planet_stats**)
- ☑ Building each unit type, showing warning messages when resources are insufficient, and subtracting the correct amounts of metal or deuterium based on unit costs
- ☑ Upgrading technologies and checking that:
  - ☑ Resource costs apply correctly
  - ☑ Levels increase properly
  - ☑ Higher levels make units more expensive
- ☑ Running full battles and checking:
  - ☑ Logs in **battle_log**
  - ☑ Battle data in **battle_stats, planet_battle_army**, and **planet_battle_defense**
- ☑ Comparing game statistics before and after battles using **printStats()**
- ☑ Implementing Java Swing interface and verifying tab behavior with info messages
- ☑ Converting battle reports from Java code → HTML → XML/XSL for users to view outside the game code

Generalitat de Catalunya
Departament d'Educació
Institut Esteve Terradas i Illa
Bonavista 70
08940 Cornellà de Llobregat
Tel. 93 377 11 00
a8016781@xtec.cat
http://www.esteveterradas.cat

Bugs Found and How They Were Solved:

Error	Description	Applied Solution
Duplicate entry a **battle_stats**	Happened when reusing the same **num_battle** without updating the battle counter	Modified **getNextBattleNumber()** to correctly read and update from the database
Foreign key violation in **battle_log**	Occurred when trying to save logs before creating a **battle_stats** entry	Added a check in **iniciarBatalla**() to ensure the **battle_stats** record exists before logging. Also updated primary keys in **battle_log**.
Nonexistent columns in **lighthunter_destroyed**	Due to mismatched class names in Java and SQL column names	Aligned class names with SQL column names and added logic to determine whether to update the army or defense tables
Duplicate entries in battle log	Log attempted to insert multiple identical lines with repeated **num_line**	Added a function **eliminarLogAnterior()** to clean up old logs before generating new ones
Inconsistency between memory state and database	Sometimes the planet had fewer resources than indicated in the database	Synced all updates so that every in-game action is immediately reflected in the database

Generalitat de Catalunya
Departament d'Educació
Institut Esteve Terradas i Illa
Bonavista 70
08940 Cornellà de Llobregat
Tel. 93 377 11 00
a8016781@xtec.cat
http://www.esteveterradas.cat

Improvements Made During Development:

Several improvements were implemented throughout the debugging and testing phase to enhance system **reliability** and **stability**, such as:

- **Persistence Modularization**: Created a **PlanetRepository** class to centralize and manage all SQL-related operations.
- **Separation of Responsibilities**: Combat logic and resource management are separated, making independent testing easier and improving code clarity.
- **Exception Handling with Try-Catch** : All database operations are wrapped in error handling (e.g., **SQLException**) to avoid game-breaking issues and show error messages instead.
- **Comprehensive Logging System**: The **battle_log** table stores all combat processes, greatly aiding in problem diagnosis and verifying that units behave correctly.
- **New Mechanics**: A new gameplay mechanic was added to:
    - Punish inactivity (if you don't act, your planet is destroyed)
    - Balance enemy and player power based on difficulty
    - Force a planet reset if you run out of both **metal** and **deuterium**

Graphic and Audio Resources

Sources Used:

The *Space Wars* project was initially developed as a **console-based application**, with an early focus on:

- Core game logic
- Data updates and mechanics

Once the core systems were in place, we integrated a **graphical interface** using **Java Swing/JFrame** to make the game more visual and interactive.

Since we added a graphical interface, we needed **images or sprites** to visually represent:

- Units
- Upgrades
- Planet states

Generalitat de Catalunya
Departament d'Educació
Institut Esteve Terradas i Illa
Bonavista 70
08940 Cornellà de Llobregat
Tel. 93 377 11 00
a8016781@xtec.cat
http://www.esteveterradas.cat

To create these assets, we used **Blender**, a 3D modeling software that's intuitive even for beginners.

We created:

- 3D models for all fleets and defense units
- Two versions of the planet model:
  - One with resources (normal state)
  - One in red (destroyed state, no resources left)

For the website, **ChatGPT** was used to generate some of the image content we used.

Integration into the Project:

To implement the **JFrame**, we used Java's built-in libraries (inside Eclipse IDE). We linked interface elements to the project using:

- **Event handling**
- Layout managers like **BoxLayour** and **BorderLayout.**

As for the **sprites made in Blender**:

- They had to be **manually imported**
- Specific functions and **absolute paths** were coded to locate the image files
- Once recognized, the images were assigned to game elements via variables and functions

Note:
At this stage, the game **does not yet include audio effects**.
However, adding sound could be a good improvement for future development.

Generalitat de Catalunya
Departament d'Educació
Institut Esteve Terradas i Illa
Bonavista 70
08940 Cornellà de Llobregat
Tel. 93 377 11 00
a8016781@xtec.cat
http://www.esteveterradas.cat

User Manual

Instructions to Run the Game:

Using the IDE:

You don't need to install anything special. Just follow these steps:

1. Download the project (as a ZIP file or clone it from the repository).
2. Unzip the file if necessary.
3. Open your preferred IDE (e.g., **Eclipse** or **Visual Studio**).
4. Navigate to the project directory and open it with the IDE.
5. Run the program — and you're ready to play!

Using the .exe File:

1. **Download the project** (as a ZIP file or clone it from the repository).
2. **Unzip** the file if needed.
3. Run the **.EXE** file of the game — and you're good to go!

Game Controls and Objective:

Planet Wars is a **turn-based strategy game** developed in Java, inspired by classics like **OGame**. In the game, you'll:

- Manage a planet
- Produce resources
- Upgrade technologies
- Build an army
- Defend against waves of enemies that attack automatically every so often

---

Your mission:
Protect your planet by improving your technological capabilities and ensuring you never run out of the two most important resources: **metal** and **deuterium**.

If you run out of either, your planet becomes vulnerable… and could be destroyed.

Generalitat de Catalunya
Departament d'Educació
Institut Esteve Terradas i Illa
Bonavista 70
08940 Cornellà de Llobregat
Tel. 93 377 11 00
a8016781@xtec.cat
http://www.esteveterradas.cat

Reflection and Improvements

Challenges Faced and How They were Overcome:

The **biggest issue** encountered during development was related to **image path management**. Each operating system (Linux, Ubuntu, Windows...) handled file paths differently, which made setting up **relative paths** quite difficult.

The problem was that IDEs interpreted file routes differently based on several factors. After multiple attempts and testing, we finally figured out how to **generalize the pathing system**, making it compatible across all operating systems.

Future Improvements and New Features:

Due to time constraints, we couldn't implement certain planned features, such as:

- **Unique planets for each player**
- **Battle creation between different player planets**
- **Multiplayer modes**

For the future, it would be a great idea to:

- Implement **new mechanics** to improve gameplay
- Take better advantage of **real-time database updates**
- Add **audio effects**, **more unit types**, and **dynamic UI changes**
- Improve **combat visualizations** for a more immersive experience